オンライン・データ・ベース・システムにおける諸問題*

穂 鷹 良 介**

Abstract

Several problems on Online Data Base Systems are discussed based on the actual experimental system BSCL 1.

Among interests of this paper are:

(1) Implementation of the garbage collection of main storage using the technique of compacting.

(2) Concurrent information storage and retrieval of online data base experimented on the system BSCL 1.

(3) Implementation of special inverted file.

(4) Introduction of system files to describe the data base.

(5) Tests of online terminals by the terminal simulator.

(6) A proposal of the new idea of the job and the transaction.

(7) Some investigations to the error recovery of the online data base system.

(8) Some investigations of resource allocation problems peculiar in online data base system circumstances.

1. はじめに

日本ソフトウエアでは昭和44年3月からオンライン IR システムを自主研究としてとりあげ，それ以来この方面の開発を進めてきたが，46年3月にその第1次試作システムを完成したのでここでその概要を報告し，さらにオンライン・データ・ベース・システム(以下 ODBS と略称する) の諸問題について論ずる.

ODBS ではオンライン・データ・ベース (以下 ODB と略称する) 特有の問題を解決しなくてはならないだけでなく，一般の multiprogramming, TSS がもつ問題もすべて問題となりうるが，ここでは話を前者だけに限りたい.

以下 **2.** ではわれわれの研究の経過について簡単に述べ，**3.** では ODBS の定義，目的について触れる. **4.** ではわれわれが試作した ODBS の一つである BSCL 1 について述べ，**5.** ではこれらの開発経験を通じて得られた ODBS の改良点，今後考えなくてはならない諸問題についてわれわれの考え方を述べる.

以下の発表でわれわれが新しく試みたことは大体次のとおりである。

(1) BSCL 1 システムで dynamic storage allocation を行なう際に，従来やってはいけないとされている領域の compacting を伴う garbage collection を行ない，そのとき要する時間を実測したこと.

(2) BSCL 1 システムで行なった ODB に対して concurrent に情報を蓄積し検索したこと.

(3) BSCL 1 システムで最終的には予定した全機能をもつには至らなかったが，索引を用いるランダム・アクセス法の開発を行なったこと.

(4) BSCL 1 システムでデータ・ベース自身を記述するシステム・ファイルの工夫を行なったこと.

(5) BSCL 1 システムで通信回線を用いずに一部端末シミュレータを用いてテストを行なったこと.

(6) ODBS で有用と思われるジョブの形態についての提案を行なったこと.

(7) ODBS で生ずるエラーの回復についての諸手段を概念的に整理したこと.

(8) ODBS の設計に際して考慮しなくてはならない資源配分についての特殊問題を指摘したこと.

2. 開発の経過[1),2)]

本研究は，はじめオンライン IR 用の言語を定める

---

* On some Problems of Online Data Base Systems, by Ryosuke Hotaka (Nippon Software Co. Ltd.)

** 日本ソフトウエア株式会社

作業に重点がおかれていた．言語設定作業は昭和44年4月電子技術総合研究所（当時は電気試験所）が提案したSCL 0[3]を母体にしてそれに各種の機能を追加することが行なわれ，44年9月にはかなり仕様の大きな言語 SCL 0.5 が定まった[1]．

実際に implement する段階になってよく反省してみると，オンラインという条件のもとにあるシステムを動かすためには，単なる言語以上にOSとのかかり合いが問題となることがわかったので，実際に作るIR用言語としてはSCL 0.5の仕様を大幅に制限したものにし，主力をオンライン関係のプログラム開発にあてることにした．

44年度後半から45年度にかけてはこのような問題に対しての意識の変化があったので，45年度のプロジェクトでは，プロジェクト名をIRだけに制限せず，データ・ベースを一般的に扱うため「オンライン・データ・ベース」とした．

45年度末にはオンライン・データ・ベースの第1次試作システム BSCL 1 (Basic Symbolic Command Language 1) を完成した．

現在は引き続いて第2次システムの設計を行なっている．

3. オンライン・データ・ベース・システムの目的

はじめにわれわれがここで用いるデータ・ベースという言葉の定義を与える．

定義　データ・ベースとはファイル化された情報のたがいに関連ある集合で，ファイルのアクセス，保守などが統一された見地から行なわれ，ファイルに蓄えられる情報相互に整合性のあるものである．

ODB とはこれらのデータ・ベースがオンラインで結合された端末からアクセス可能になっているものをいう．

ODBSの目的は，オンラインの環境のもとで以下のようなデータ・ベースに対してのサービスを提供することにある．

（1）　データ・ベースにアクセスする標準的な手段を提供することにより，統一された方法で個々のファイルの maintenance が行なわれる．

（2）　ユーザはファイルに対してのアクセスをODBSが提供する手段を通じて行なう結果，計算機のオペレーティング・システムのデータ管理方法の変更，物理的な装置の変更が生じても，プログラムの変更はしなくてもすむ．

（3）　複数個のアプリケーションで互いに共有するデータはデータ・ベース中1箇所におくことによって，その更新・保守を矛盾なく行なう．

（4）　オンライン・モードでもバッチ・モードでも同様にデータ処理ができる機能を提供する．

（5）　エラー回復など非常に手数のかかる処理に対して標準的でしかも強力な機能を提供する．

（6）　プログラムの開発費用，専用記憶領域，処理時間の軽減をもたらす．

4. BSCL 1 システム

4.1 システムの概要

BSCL 1 は FACOM 230-60に対して開発された．concurrent に動く端末の数は4個としておのおのにタスクが1個対応する．これらのタスクを発生させ，動作中に割込みをかけて停止，再開などの制御を行なうのがマスタ・タスクであり，これらの合計5個のタスクがオペレーティング・システムのもとに非同期に動作する（**Fig. 1** 参照）．

端末に対応する4個のタスクのうち2個は端末シミュレータとして実現されており，実際 active に動く端末はタスク3，タスク4の2個である．

マスタ・タスクはコンソール・タイプライタと情報の交換を行ない，オペレータの指示を受けるほか，適当なメッセージを出力する．

タスク間の制御の切り換えは，WAIT*1が出された

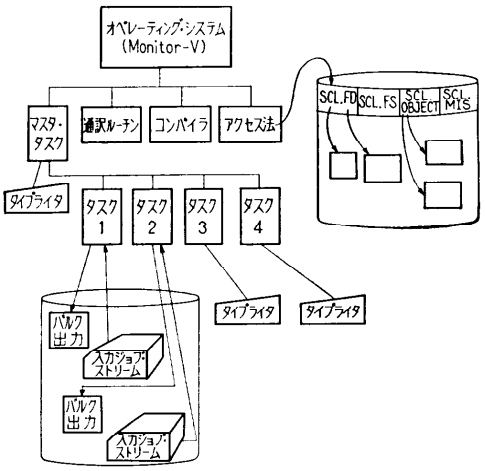


Fig. 1　general construction of BSCL 1

*1 入出力命令の終了，その他の event の生ずるのを待つシステム・マクロ命令

ときなどオペレーティング・システムによってなされる．各端末からは BSCL 1 の文法にしたがったプログラムを投入することができ，それらがコンパイラによってコンパイルされてテーブル形のオブジェクト・プログラムに変換されたものを通訳ルーチンが 1 step ずつ解釈実行する．この意味では，この通訳ルーチンは run time subroutine の性格が強い．

一度コンパイルされたプログラムはこれをシステムファイルに登録しておき，各端末で共通に使用することができるため，システム・プログラムだけでなく，ユーザ・プログラムもリエントラントになるように翻訳される．

システムの大きさはプログラム部分だけで 45 kW ほどあるが，最適化すればもう少し減る可能性がある．システムがシステム・ファイルにアクセスする場合，実行時にユーザ・プログラムがデータ・ベースにアクセスする場合，その他いかなる場合でもデータ・ベースにアクセスするときには BSCL 1 に備わっているアクセス法（詳細は **4.4** データ管理の項を参照のこと）を使用することにより，データ・ベースの統合性を保つようになっている．

端末シミュレータの構造は次のようになっている．シミュレートする端末ごとにカード・イメージの入力ジョブ・ストリームをあらかじめ用意しておいて，端末からの入力を受け付けるのと同じロジックでシステムが 1 レコードずつソース・プログラムまたは入力データを読み込む．

端末シミュレータに対しての出力も端末ごとに設けられたファイル相手になされ，この内容は BSCL 1 全体のジョブが終了したときにバルク出力の形でユーザに一括返される．

4.2 タスク管理

BSCL 1 は，全体としては MONITOR-V*[2] のもとに動く 1 つのジョブとして動作するが，オペレーティング・システムから 1 度コントロールを受けた後，端末の数だけタスクを発生させるので，それら端末で走るジョブに対するタスク管理を行なう必要がある．具体的には次の 3 つのことを行なう．

（1） 各端末ごとに実行されるプログラムは，他の端末ジョブによって同時に実行可能なようにすべてリエントラントになるようにする．

（2） データ・ベースを端末間で共同に使用することがある以上，システムに属する各種の資源を異なる端末ジョブが同時に競合して用いる可能性があるので，目的に応じて排他的な制御ができるようにする．

（3） **4.3** で述べる主記憶の garbage collection のため全端末ジョブを同期させて一時止め，必要な処理を行なった後に再開させる．

これらの点について更に説明しよう．

（1）は，**4.3** の主記憶管理ルーチンの機能を用い，端末ジョブが必要とする主記憶を実行時にそのつど割り当てる方式によって実現している．

（2）は，制御モードを 2 つ設けることによって行なっている．1 つはある種の処理を行なうのにただ 1 個のタスクにしか制御権を渡さないようにするモードである．このような制御はしばしば情報の書き込み中に必要とされるので，制御権をつかむという意味をこめて，seize write モードと呼ぶ．もう 1 つは処理を行なうのに複数個のタスクに制御権を渡すことは認めるが，どれか 1 つのタスクで seize write モードで同時に処理をすることは認めないというモードで，しばしば情報をただ読み取るという動作に必要とされるので seize read モードと呼ぶ．この 2 つの概念は，BSCL 1 の開発途中で必要にせまられて導入したものであるが，あとで調べたところ，IBM の OS/360 での ENQ E (E は exclusive の意) と ENQ S (S は shared の意味) という概念と全く同質であることがわかった．なお，seize read の概念は MONITOR-V にはないが，seize write の機能を使えば簡単に作ることができる．

（3）勝手に走っている端末ジョブの同期をとって一斉に停止させるということは，そんなに簡単ではない．BSCL 1 では大体次のような方法を取った．

（i） 端末ジョブを止めるという指令はコンソール・タイプライタに対応しているメイン・タスクを経て出される．

（ii） 各端末ジョブは一定時間間隔内に少なくとも一度メイン・タスクから停止の指令が出ていないかどうかをチェックする（実際は，頻繁に通過する共通ルーチンの中にこのチェックを入れておく．）．

（iii） 停止指令を受けとった端末ジョブからは停止指令を受けとったという了解の信号をメイン・タスクに送り，自分は停止する（WAIT 状態にはいる．）．

以上のロジックは大まかなもので，実はこれだけでは端末ジョブが停止指令を出す以前から停止の状態に

*[2] FACOM 230-60 のオペレーティング・システムの 1 つ．

ある場合や，現在は走っているが，(ii)で述べたチェックにはいらずにそのまま停止する予定の端末からは(iii)の応答を得ることができなくて，メイン・タスクは永遠に待つはめになってしまう．したがって次のような修正が必要となる．

(iv) すべての端末ジョブのすべての停止状態がメイン・タスクの方で把握できるようにしておいて，現在停止している端末ジョブ，およびチェックをしないで停止に至る端末が強制的にチェックにはいるのと同じ効果をもつようにする．BSCL 1 ではこれをチェックによって停止に至る端末ジョブに対してもダミーの応答をさせるような方式と，WAIT が生ずる場所を各端末ごとに1箇所にすることによって解決している．

4.3 主記憶管理

プログラムをリエントラントにするためにdynamic storage allocation の方式を利用することにしたことはすでに **4.2** で述べたとおりであるが，MONITOR-V にわれわれの目的に合うような小単位の storage allocation の機能がないので，BSCL 1 自身で dynamic storage allocation を行なうことにした．機能と特徴は次のとおりである．

(1) BSCL 1 システムは，はじめに MOMITOR-Vから大きな単位で主記憶を獲得する．

(2) その後端末ジョブが走る際に要求する主記憶を variable length で分け与える．

(3) 不要になった主記憶は端末ジョブから随時返却を受付る．

(4) アルゴリズムは本質的には first-fit[4] の方法で行なうが，領域の返却に伴って生ずる利用できない主記憶のくずを集めるため compacting を伴う garbage collection を実行する．compacting を行なうときにはある端末が動いているというようなことがあっては，その端末が不正な情報を使用するおそれがあるので，すでに **4.2** で述べた方法によって全端末をいったん停止させ，garbage collection を行なった後にベース・レジスタの set のし直しをして再びコントロールを端末にもどす．

一般に compacting を伴う garbage collection は，時間がかかりすぎるという理由のために嫌われているようであるが，高速磁心記憶上に 200 バイトの領域が 163 個，低速磁心記憶上に 200 バイトの領域が 655 個開設されているとき一番時間のかかるやり方で garbage collection を行なって，約2.6秒要した．

なお，garbage collection に伴って全タスクの動きを止めるやり方は，OS の機能がアップすればもっとうまいやり方があろうかと思われるが，現在の OS ではそこまでこちらの微妙な使い方に即応できる機能をもっていないように思われる．このテクニックは，あとで述べる throb point での情報収集の際にも必要となると思われる．

4.4 データ管理

ランダム・ファイルとしては下に述べるような構造をもつファイルを開発しようとして，作業は途中まで進んだが，下の(1)に述べるような困難があって一部未完成に終わった．

ここで簡単にわれわれが設計したファイル構造の概要を **Fig. 2** を使って説明する．

BSCL 1 のデータ・ベースに登録されるファイルはすべてシステム・ファイル SCL. FD にアクセスに必要な情報がはいっているので，ユーザはファイル名を指定することによって目的とするファイルを処理することができる．

ランダム・ファイルは単数レコード型と複数レコード型に分類される．おのおのはレコードあるいはレコード群に一意に対応するキーをもち，このキーについての索引構造によって，単数レコード型のときにはレコードそのもの，複数レコード型のときにはレコード群の最初の BOR (Beginning Of Records) にたどりつくことができる．複数レコード型のときには BOR を検出したところからは sequential にレコードを処理することができ，レコード群の終わりは EOR (End Of Records) によって検出できる．インバーテッド・ファイルはこの複数レコード型の機能を用い，他の単数レコード型のファイルのキーを蓄えることにより，作成することができると思われるが，実際に作成はし

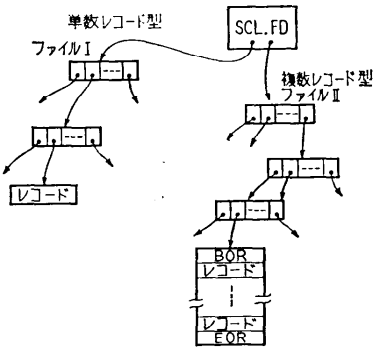


Fig. 2 file structure of BSCL 1

```
*** ... S C L 0.5 ... START TERM. NO. = 3 ***
<SCL0.5>
0001
0002  (PROC)EISFILE
0003  DCL   FDA UNIT,2(FLNAME CHAR(12),RES CHAR(11A),RTCODE,FLAG,DTLEN,
0004  KEY CHAR(20));
0005  MOVE 'EISFILE     ' TO FLNAME
0006  MOVE 0 TO FLAG
0007  OPEN (INPUT FDA=FDA
0008  IN: TYPE IN KEY
0009        (KEY='12345678901234567890', GOTO END)
0010  IN FDA=FDA
0011  TYPE OUT EISFILE_ENG
0012  TYPE OUT EISFILE_FREN
0013  TYPE OUT EISFILE_GERM
0014  GOTO IN
0015  END: END
*** ... S C L 0.5 ... END TERM. NO. = 3 ***
```

2行目	(PROC)	プログラムの宣言.	9行目	(C, S)	条件 c が true のとき s を実行する.
3行目	DCL	変数の宣言.	10行目	IN	入力命令.
5行目	MOVE	データ移動命令.	11行目	TYPE OUT	端末への出力命令.
7行目	OPEN	ファイルのオープン命令.	14行目	GOTO	ジャンプ命令.
8行目	TYPE IN	キーボードからの読み込み命令.	15行目	END	ソース・プログラムの終了宣言.

Fig. 3 an example and the explanation of a sample BSCL 1 program

なかった.

そのほか, データ管理については (2), (3) のような問題がある.

(1) 索引をもつファイルでありながら, その追加・変更を無制限に許す方式にしておいたため, オーバ・フロー, 領域のむだづかい, それに伴うアクセス時間の増加が予想され, さらにこれらをすべてカバーするロジックは非常に複雑となった. アクセスを早くしようとするファイル構造では, 実用的な見地からはオンラインの更新を許すのはかなりむずかしいことがわかった.

(2) ファイルを更新しようとする場合にはもちろん, 単に検索をしようとするときでも該当索引部分を主記憶においてデータ処理をする必要があるので, 実質的に排他的な制御(seize write) モードになる場合がある.

seize write モードが多く使われる場合には, それだけ資源の利用が制限されるわけで, これを少なくする方法を考える必要がある.

(3) ODB の中で最も重要な問題の1つに機密保護の問題がある. BSCL 1 ではこれをファイルごとレコードごとフィールドごとに合言葉をもっていて, それが符合しないときにはユーザに対応する部分のデータを渡さないようになっている*3. しかし合言葉は機密保護の完全な解決にはほど遠いものであって, これを完全に解決するためには別のハードウェアによるユーザの識別方法がなくてはならないと思われる.

4.5 システム・ファイル

BSCL 1 には SCL.FD, SCL.FS, SCL.OBJECT, SCL. MIS という4つのシステム・ファイルが設けられていてデータ・ベースの管理に用いられている. なお, これらのファイルは, すべて **4.4** で述べたランダム・ファイルの構成をとっている.

(1) SCL.FS データ・ベース中に存在するすべてのユーザ・ファイルについての file structure の定義を登録している.

(2) SCL.FD ユーザ・ファイルについての記述の内, SCL. FS に盛られなかった情報をすべて記録する. physical location などもはいっている.

(3) SCL.OBJECT BSCL 1 コンパイラによってコンパイルされたオブジェクト・プログラムを登録する.

(4) SCL.MIS データ・ベース全体に関する情報, たとえばディスクのシリンダの使用状況など, 雑多な情報 (Miscellaneous Information) を蓄えておく.

4.6 言 語

BSCL 1 のプログラム言語は SCL 0.5 のサブセットということになっているが, 実際に implement する段階で多少変更が加えられた. またこの言語を使って実際に端末からプログラムを打ち込んでみると, 使いにくい点がいろいろと現われ, 反省の材料を得ること

*3 ファイルのオープン時に一度だけファイル, レコード, フィールドについての合言葉の一致不一致が確かめられ, ファイルについての合言葉が合わないときには, ファイルの全内容, レコードについての合言葉が合わないときには, レコードを読み出したときにさらにそのレコードについている秘密保護の指定の有無を見て保護する必要のあるときには, そのレコードの内容をユーザに渡さない, フィールドについては合言葉が合わなかったときには, レコードを読み出した後, 対応するフィールドを同じ文字でクリアしてからユーザに渡す方式をとっている.

ができた．**Fig. 3** に例を示す．

BSCL 1 で使用できる命令は，宣言文1種，ファイル定義命令2種，入出力命令6種，制御命令2種，データ転送命令1種，計算命令1種となっている．

5. ODBS の諸問題

5.1 ジョブとトランザクション

ODBS に限らず一般に multiprogramming をやっているときにはタスク間に資源のうばいあいが生ずる．資源としてはデバイス，ファイル，主記憶，プログラムなど，一つのタスクで使用することが他のタスクにその資源の利用をすこしでも妨げる可能性のあるものすべてが対象となる．

資源の割当ては慎重に行なわないと deadlock を生ずるおそれがあるが，あまり慎重にやると全体で走っているタスクが1つだけというような効率の悪い使い方になるおそれもある．

そこで資源利用を高める1つの方法として，1つのジョブを走らせるときに資源としては最低限必要なものだけの割当てをジョブのはじめに受けておいて，それ以上の資源は，実行時に必要に応じて割当てを受け，不用になったら即座に返却するということが考えられる．もちろん，実行時に要求する資源がタスク間あるいは端末間で競合して deadlock になる可能性もあるが，そのときには deadlock を検出するロジックを組み込んでおいて，再度要求させるかあるいは以後の判断をユーザにまかせることにする．ここで注意すべきことは，実行時に資源を要求してそれに対してデータ処理をする一連の動作は，資源要求の段階で途中不成功に終わることもあるから，再度試行してもかまわないという性質のものでなくてはならないことである．

このようにジョブの実行中に新たに資源を要求してそれに対して再試行可能なデータ処理を施す処理単位をわれわれはトランザクションと呼ぶことにする*4．

トランザクションの定義から結論されることであるが，トランザクションの実行中はそのタスクがコントロールを独占しないと悪影響をもつことになるであろう．トランザクションをどの程度のものまでとするかはアプリケーションによっていろいろとやり方があるであろうが，1トランザクションであまり複雑な入出力処理をするのはこの意味で避けなくてはいけない．

*4 ここで定義しているトランザクションは現段階ではまだ机上の一設計概念にすぎなく，その正確な定義づけ，実現のための設計方法については後日稿を改めて再述したい．

何も資源を要求しないデータ処理も1種のトランザクションとみたてて不都合はないから，ジョブはトランザクションの列から構成されると考えてよい．

ジョブとは ODBS の場合，端末からユーザが実行すり一連のデータ処理で，ロジック的に一区切りとなるものをいう．

ジョブとトランザクションは，以下でも見るようにエラー回復の単位にもなる．

5.2 エラー回復のための準備

ODBS ではエラー回復が最も重要でかつむずかしい問題の1つになるであろう．ODBS でもオンラインで ODB に変更が加えられる場合とそうでない場合とでは回復の手段の量と質とが著しく異なる．変更が加えられない場合には，現在使用されているデータ・ベースのバック・アップを常にとっておくことによって致命的なエラーはかなり避けることができる．その場合には，通信回線関係の通常のエラー処理と，バッチ処理で採用しているエラー処理を用意するだけで話はすむ．

これに対して，オンライン更新を行なう場合には時々刻々とデータ・ベースの内容が変化していき，しかもその変化のたびごとにバック・アップをとるということは不可能に等しいので問題はおそろしく複雑になる．筆者のラフな見積りであるが，オンライン更新を行なわない場合と行なう場合とではプログラミングに要するマンパワーにして5～10倍の差があろうかと思われる．

ここではこのオンライン更新を行なう際に，エラー回復のために必要とする準備について述べ，**5.3** で回復方法についての一方法を示す．

(1) throb point　システムの運行中に一定時間隔ごとに割込みをかけ，その時点でのシステムの状況をログ・ファイルに蓄える．ログ・ファイルはあらかじめ read write check を行なった磁気テープ上に作成する．このとき収集する情報としては次のようなものがある．

(i) ファイル使用状況，
(ii) 実行中のジョブとその種類，
(iii) レジスタ類とその他のシステムの制御情報．

(2) user journal　ユーザの指定によって必要な情報をログ・ファイルに書き込む．

(3) antimage, postimage　データ・ベースに対して更新のトランザクションが実行されたとき，書き

かえた情報単位にデータ・ベースの書きかえ以前の内容を antimage とし，書きかえ後の内容を postimage としてログ・ファイルに書き込む．

（4） event logging　トランザクションも含めて，システムになんらかの変化を与えた事象 (event) の記録をログ・ファイルにとる．

（5） break point back up　日単位，週単位，月単位あるいは必要に応じて適当な break point を設け，バッチ処理によってデータ・ベースのバック・アップをとる．

（6） ファイル構成　後に述べるエラー回復の手順が実行しやすいファイル構成をとらなくてはいけない．更新を伴う ODB では，単に利用という面からでなくもう一つエラー修正の容易さという判断基準からデータ構造が吟味されなくてはならない．

（7） 情報の識別方法の標準化　antimage, postimage などをログ・ファイルに書き出すためにはデータ・ベース中のどの部分の情報かということを定める識別方法が標準化されていなくてはならない．

（8） コピー機能　実行時にユーザの指令などにより，データ・ベース中の必要なファイルをコピーする機能が必要である．

5.3 で述べるようにオンライン・データ・ベースの更新を行なうためには上記で述べた程度の用心をしてかからなければならない．特に(1), (2), (3), (4) は頻繁に生ずるものであるから，そのためシステムの効率が低下することが予想される．これらは皆，ODB 更新のための高価な犠牲であって，逆にいえば，これだけの犠牲に見合うだけの merit がなければ，ODB 更新はやめて，ODB は検索のみ，更新はバッチ処理によるとすべきである．後者の場合にはエラーによって予想される困難はほとんどないように思われる．

5.3 エラー処理ルーチン

ここでは一般的にエラーが生じたときにシステムがとる手段をはじめに述べ，次にエラーの深さに応じて6段階のエラー回復手段を提示する．

通常エラーが検出されたときのシステムの反応は次のようになる．

（1） 以後エラー回復のために支障となる新規のトランザクションの受付けを禁止する．

（2） 入力中のトランザクションをすっかり入力し終わる．

（3） エラー・メッセージを出す．

（4） 以下に述べるエラー回復手段を順に試みる．

（5） エラーがうまく回復したときにはエラー処理ルーチンにはいったため未処理のまま残された諸手続きを行ない，再びトランザクションの受付けを開始する．

6段階のエラー回復手順は次のようである．

〔第1段階〕　retry 可能なトランザクションのエラーの場合には retry を何回か試みる．

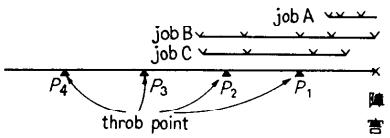
〔第2段階〕　新たな変化がシステムに対して生じないようにシステムを閉鎖した後，ODB の状態を最後の throb point にまでもどして，そこから処理を再開する．

ODB の状態を時間的に以前の状態にもどすには，**5.2** の(3)で用意した antimage の情報を逆に読むことによってなされる．この際，その間にはいってきたトランザクションと antimage, postimage の間に矛盾がないかどうかをチェックしながらもどる．**Fig.4** でいえば×印のところで障害が起きたとすると throb point P1 まで ODB の状態をもどすのである．このときその間に生じた ODB への更新トランザクションだけをトレースする．throb point P1 にもどったところで，計算機の内容を P1 で出力したときの内容と等しくして，今後は postimage を用いて処理を再現していく．

〔第3段階〕　このエラー回復方法はジョブ単位にエラーの回復を試みるものである．Fig. 4 で障害の原因となったジョブがAだけであるとすると，第2段階と同様の方法で throb point P1 までもどり，そこから，ジョブAに関連したトランザクションの処理だけを再現する．

このとき，エラーの原因がジョブAだけであると断定することができないならば，その時点で完了していないジョブ (Fig. 4 ではジョブAとB) についてその起源にもどり (Fig. 4 では throb point P3 までもどる) そこから，ジョブAとBとに関連したトランザクションの処理を行なう．

〔第4段階〕　適当な break point でとった ODB



v 印はトランザクションを示す

Fig. 4　relations of throb point, job, and the transaction

の back up から ODB を再現し，その時点から生じたすべてのトランザクションを再実行する．

〔第5段階〕　ユーザが特別に用意する break point にもどり，ユーザの提供するエラー回復手順を実行する．

〔第6段階〕　システムの現在の様子をメモリ・ダンプしダウンする．回復は人力によって行なう．

オンライン更新を行なう ODBS の開発が更新を行なわないシステムに比較してマンパワーを余計に必要とするのは，おもにここで述べたエラー回復を行なうための準備プログラムのためである．

また**5.2**で述べたようなログ・ファイルをシステムの走行中に常時作成していくためにはかなりの費用がかかる．オンライン更新を伴う ODBS を導入するためには，これらの覚悟をしておく必要がある．

5.4 資源の配分

ODBS では一般のオペレーティング・システムで行なわれているやり方よりも，もっときめ細かい資源配分方法が望まれる．

たとえば一つのデータ処理において，あるファイルの内容が一つのジョブによって更新され始めたら，一切ほかのジョブによって同じファイルを更新させたくないという場合もあろうし，レコード単位，ビット単位で更新がかちあわなければ実用に差し支えないという場合もある．

このような処理情報の精粗のほかに，ジョブまたはトランザクションがこの資源を保有する時間の長さも問題になる．たとえば1レコードを更新する間だけ資源を保有していればよい場合，一つのファイルについて OPEN から CLOSE までの間保有したい場合，1ジョブの間保有しておきたい場合と様々なケースが生れる．

BSCL 1 ではファイルの処理モードを制御モードと同じく

(1) read only (読むだけ).

(2) exclusive (自分以外のものは触れるのもいけない).

の2つにしかわけなかったが，もっと資源の利用状態をふやしてたとえば

(3) read while changing (現在書きかえられているものでもかまわず読んでよい).

(4) protect against current updates (現在扱っている情報を自分以外のものが書きかえさえしなければ読んでも差し支えない).

(5) copy (一度コピーしたファイルに対して処理を行なう).

などというモードを追加することが考えられる．以上の資源の精粗，保有の長さ，モードはいずれも独立に変化しうると考えられるから，ファイル資源の配分形態は非常に多くなる．

また ODBS では資源の使用に対して課金を行なわないと，ユーザが過大な資源の要求をする結果，システムの効率低下を招くおそれがあるから，課金ルーチンを設ける必要があるが，価格のきめ方，課金のきめ方などは上の資源の配分形態を反映して一般の OS の会計ルーチンよりもはるかに複雑になることが予想される．

6. おわりに

本研究は，日本ソフトウエア開発2部，鷹木真敬，小林昌之，鳩宿　憲，森田勝弘の諸氏の協力のもとになされた．ここに感謝の意を表したい．

参考文献

1) 日本ソフトウエア株式会社：「SCL 0.5-3 外部仕様書」，昭和44年9月.
2) 穂鷹良介：「オンラインIR システムSCL 0.5」，昭和44年度，情報処理学会大会講演予稿集, 95～96.
3) 五十嵐実子：「記号化された問い合わせ言語の文法」，昭和44年度情報処理学会大会講演予稿集，89～90.
4) D. E. Knuth: The Art of Computer Programming Vol. 1, Fundamental Algorithms, Addison-Wesley, 1968.
5) 森岡　武，吉田達明，加藤栄護：「加入データ通信販売在庫管理システム(Ⅶ)，障害復旧について(1)」，昭和45年度第11回大会講演予稿集，情報処理学会 145～146.
6) 森岡　武，野々市谷建，加藤栄護：「加入データ通信販売在庫管理システム(Ⅷ)，障害復旧について(2)」，昭和45年度第11回大会講演予稿集，情報処理学会，147～148.
7) 石川　宏，垣原正紀，橋本健一郎，三井正征：「DEMOS 障害対策プログラム」，上記予稿集，237～238.
8) 関　栄四郎，小口和郎，中野　勝，安助秀一，角野正行：「国鉄貨物情報処理システムの障害回復」，上記予稿集，331～332.
9) IBM Information Management System/360 for the IBM System/360 (System Description) Application Manual, H 20-0524-1.
10) CODASYL Systems Committee, Technical Report: "A Survey of Generalized Data Base Management Systems" May, 1969.
11) David Hsiao, Frank Haray: "A Formal System for Information Retrieval from Files", CACM 13, 2 Feb., 1970, 67～73.

(昭和46年5月6日受付)